BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1. Tinjauan Pustaka

Beberapa penelitian/jurnal yang terkait dengan penelitian tugas akhir ini adalah :

.

NO	Peneliti	Judul	Teknologi/ Metode	Fungsionalitas
1	Bella Hardiyana, S.Kom, M.Kom dan Irfan Suendi, Amd.Kom	Sistem Informasi Pendataan Bayi (Studi Kasus) : Posyandu Dahlia RW/RT12/05 Kec.Baleendah, Kel.Baleendah, Kab.Bandung)	*Java, MySQL*	Pengolahan data bayi di posyandu,dan pembuatan laporan.
2	Denty Monika Sales Rachmansyah, S.Kom (2013)	Sistem Pengelolaan dan Pemantauan Posyandu Berbasis Web di Kota Palembang	*Rational Unified Process (RUP), Unified Modelling Language (UML) ,PHP, Adobe Dreamweaver , MySQL.*	Mengetahui informasi mengenai Posyandu yang ada di kota Palembang.
3	Indra Setyarini (2016)	Perancangan Sistem Informasi Posyandu Guna Mendukung Pelaporan Data Perkembangan Bayi Dan Balita	Delphi, *MYSQL*	Mempermudah menginputkan data bayi dan balita,data laporan perkembangan bayi dan balita dalam bentuk grafik.
4	Resmon Frima1, Budhi	Perancangan dan Implementasi	*Java, MySQL*	Pengelolaan database balita pada

	Irawan2, Burhanuddin Dirgantoro 3 (2016)	Sistem Informasi Posyandu Terintegrasi Berbasis Android		Posyandu, memberikan perhitungan nilai gizi balita lebih akurat.
5	Penelitian Saat ini (2020)	Layanan Informasi Posyandu Berbasis Android	*Java, PHP, Adobe Dreamweaver, MySQL*	Informasi Layanan Posyandu, Pengolahan data anggota Posyandu.

2.2. Dasar Teori

2.2.1. Sistem

"Sistem adalah kumpulan dari komponen-komponen yang saling berhubungan yang saling berinteraksi untuk melakukan suatu tugas untuk mencapai suatu tujuan". Sistem adalah kumpulan komponen yang saling berhubungan dengan batasan yang jelas, dan bekerja sama untuk mencapai tujuan dengan menerima *input* dan menghasilkan *output* dalam suatu proses transformasi yang terorganisasi. Dalam sistem terdapat 3 komponen dasar yang terdapat didalamnya ,seperti (Jogianto, 2005) :

1. *Input*, memasukkan elemen-elemen (data mentah) yang akan diproses.
2. *Process*, proses transformasi *input* menjadi *output.*
3. *Output*, mengirimkan elemen-elemen (data mentah) yang telah diproses ke tujuannya.

Jadi, sistem adalah sekumpulan komponen yang saling terkait dan bekerja sama melakukan suatu tugas untuk mencapai suatu tujuan.

2.2.2. Aplikasi Mobile

Aplikasi adalah *software* yang dibuat oleh suatu perusahaan komputer untuk mengerjakan tugas-tugas tertentu, misalnya Microsoft Word dan Microsoft Excel (Dhanta, 2009).

Kata *mobile* mempunyai arti bergerak atau berpindah. Sehingga diperoleh pengertian bahwa aplikasi *mobile* merupakan aplikasi yang dapat dijalankan walaupun pengguna berpindah atau karena pengguna berpindah (Darytamo, 2007). Pemrograman aplikasi *mobile* tidak banyak berbeda dengan pemrograman konvensional pada PC. Aspek karakteristik dari perangkat bergerak sering mempengaruhi arsitektur dan implementasi dari aplikasi tersebut. Dalam pemrograman aplikasi bergerak berbagai aspek teknis perangkat lebih menonjol karena memiliki banyak keterbatasan dibandingkan komputer konvensional atau PC.

2.2.3. Basis Data

Basis data terdiri dari 2 kata, yaitu basis dan data (Fathansyah, 2002). Basis dapat diartikan sebagai markas atau gudang, tempat bersarang atau berkumpul. Sedangkan Data adalah representasi fakta dunia nyata yang mewakili suatu obyek seperti manusia, barang, peristiwa, konsep, keadaan dan sebagainya.

Basis data juga dapat didefinisikan dalam beberapa sudut pandang seperti berikut (Fathansyah, 2002) :

1. Himpunan kelompok data yang saling berhubungan yang diorganisasikan sedemikian rupa sehingga dapat dimanfaatkan

kembali.

2. Kumpulan *file/*arsip yang saling berhubungan yang disimpan dalam media penyimpanan elektronis.
3. Kumpulan data yanng saling terhubung yang disimpan secara bersama sedemikian rupa dan tanpa pengulangan (redundansi) yang tidak perlu, untuk memenuhi berbagai kebutuhan.

2.2.4. Unified Modelling Language (UML)

Unified Modeling Language (UML) adalah sebuah bahasa pemodelan yang telah menjadi standar dalam industri *software* untuk visualisasi, merancang, dan mendokumentasikan sistem perangkat lunak (Turck, 2007). Bahasa Pemodelan UML lebih cocok untuk pembuatan perangkat lunak dalam bahasa pemrograman berorientasi objek (C# , Java, VB.NET), namun tetap dapat digunakan pada bahasa pemrograman prosedural.

Unified Modeling Language (UML) biasa digunakan untuk (Henderi, 2007) :

1. Menggambarkan batasan sistem dan fungsi-fungsi sistem secara umum, dibuat dengan *use case* dan *actor.*
2. Menggambarkan kegiatan atau proses bisnis yang dilaksanakan secara umum, dibuat dengan *interaction diagrams.*
3. Menggambarkan representasi struktur statik sebuah sistem dalam bentuk c*lass diagrams*. Membuat model *behavior* yang menggambarkan kebiasaan atau sifat sebuah sistem dengan *state*

transition diagrams.

4. Menyatakan arsitektur implementasi fisik menggunakan *component and development diagrams.*
5. Menyampaikan atau memperluas *fungsionality* dengan *stereotypes* (Turck, 2007).

UML lahir dari penggabungan banyak bahasa permodelan grafis berorientasi objek yang berkembang pesat pada akhir 1980-an dan awal 1990-an. Sejak kehadirannya pada tahun 1997, UML menghancurkan menara Babel tersebut menjadi sejarah (Fowler, 2004).

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan sintak/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak berbasis o*bject-oriented*.

A. Diagram UML

UML menawarkan *standard* untuk merancang sebuah sistem. Diagram- diagram dalam UML terdiri dari diagram *use case*, diagram *activity*, diagram *class* dan diagram *sequence*.

1. Diagram *use case*

   Dalam konteks UML, tahap konseptualisasi dilakukan dengan pembuatan *use case* diagram yang sesungguhnya merupakan deskripsi dari bagaimana perangkat lunak akan digunakan oleh penggunanya (Nugroho, 2009).

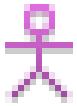
   Use case diagram merupakan gambaran *graphical* dari beberapa

atau semua *actor, use case,* dan interaksi diantara komponen-komponen tersebut yang memperkenalkan suatu sistem yang akan dibangun. *Use case diagram* menjelaskan manfaat suatu sistem jika dilihat menurut pandangan orang yang berada di luar sistem. Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

Dalam sebuah model perancangan dimungkinkan terdapat satu atau lebih *use case* diagram. Dalam tahap analisis *use case* diagram untuk menangkap kebutuhan sistem dan memahami bagaimana seharusnya sistem bekerja. Notasi yang digunakan dalam membangun diagram *use case* antara lain dapat dilihat pada tabel 2.2.

Tabel 2.1 Notasi Diagram *Use Case*

NO	GAMBAR	NAMA	KETERANGAN
1		*Use Case*	Menjelaskan apa yang sistem lakukan, bukan bagaimana sistem berjalan. Sebuah *use case* mengandung banyak skenario, dimana masing-masing menjelaskan aliran yang lebih spesifik dari *use case*. Skenario di dalam *use case* harus dapat dijelaskan dengan cukup jelas agar pihak awam dapat mengerti.
2		*Package*	Sebuah pengelompokan dari model-model elemen. Packages digunakan didalam diagram *use case*, diagram *class* dan diagram component. Sebuah *package* dapat berada di *package* lainnya, dan mengandung subordinate *packages* dan model elemen biasa. *Package* dapat berupa subsistem dari model. Subsistem secara keseluruhan dapat diartikan sebagai package, dan subsistem dengan level tinggi mengandung semua elemen pemodelan sistem.

3		*Actor*	Menjelaskan bagaimana *user* berinteraksi dengan *use case*. Aktor dapat berupa apapun seperti : manusia, *device*, ataupun sistem lain. Aktor dapat mempunyai attributes dan operations. Sebuah objek fisik dapat memainkan beberapa peran dan dapat diperankan oleh beberapa aktor.

Tabel 2.2 Lanjutan Notasi Diagram *Use Case*

NO	GAMBAR	NAMA	KETERANGAN
4		*Include*	Menjelaskan sebuah ketergantungan diantara *use case* dimana aturan yang terdapat pada suatu *use case* secara jelas menjadi pedoman pada *use case* yang lain.
5		*Extend*	Sebuah ketergantungan antara *use case* dimana aturan pada suatu *use case* dapat diperluas lagi.
6		*Generalization*	*Generalization* adalah sebuah hubungan taksonomi antara elemen umum dengan elemen yang lebih spesifik yang konsisten terhadap elemen pertama dan menambahkan informasi aditional. Hubungan ini biasa disebut spesialisasi atau hubungan *inheritance*.
7		*Association*	Hubungan yang mengasosiasikan sebuah aktor ke *use case*. Hubungan ini menunjukkan keikutsertaan aktor didalam *use case* dan hanya *link* ini yang menjadi satu-satunya penghubung antara aktor dengan *use case*.

2. Diagram *activity*

Sebuah diagram *activity* hanyalah sebuah diagram alur kerja yang menggambarkan berbagai pengguna kegiatan, orang yang melakukan aktifitas masing-masing, dan aliran sekuensial kegiatan (Satzinger, 2005). Berikut adalah beberapa notasi yang dipergunakan dalam diagram *activity* terlihat dalam tabel 2.4.

Tabel 2.3 Notasi Diagram *Activity*

NO	GAMBAR	NAMA	KETERANGAN
1		*Invocation*	Penggunaan sebuah perilaku dalam sebuah aktifitas.
2		*Activity Edge*	Sebuah kelas abstrak untuk koneksi langsung antara dua notasi *activity*
3		*Activity Group*	Sebuah invocation node dan tepian aktifitas dalam sebuah *activity*. Sebuah pengelompokkan umum untuk notasi dan tepian.
4		*Initial Node*	Sebuah notasi kontrol yang merupakan sebuah awal dari aliran aktifitas data.
5		*Activity Final Node*	Sebuah notasi akhir yang menghentikan semua aliran aktifitas data.
6		*Flow Final*	Sebuah notasi akhir yang mengakhiri aliran dan membunuh semua token yang datang.
7		*Decision*	Sebuah notasi kontrol untuk memilih salah satu aliran.

Tabel 2.4 Lanjutan notasi Diagram *Activity*

NO	GAMBAR	NAMA	KETERANGAN
8		*Vertical Fork*	Sebuah notasi kontrol vertikal yang membagi aliran menjadi banyak aliran yang terjadi dalam satu waktu.
9		*Horizontal Fork*	Sebuah notasi kontrol horizontal yang membagi aliran menjadi banyak aliran yang terjadi dalam satu waktu

3. Diagram *class*

Diagram *class* adalah himpunan kelas-kelas beserta hubungan/relasi/asosiasi antarkelas (Nugroho, 2009). Diagram *class*

menggambarkan keadaan (atribut/*property*) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut. Diagram *class* menggambarkan struktur dan deskripsi *class*, package, dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. Notasi yang digunakan dalam membangun. Notasi yang digunakan dalam membangun *diagram class* antara lain dapat dilihat pada tabel 2.6.

Tabel 2.5 Notasi Diagram *Class*

NO	GAMBAR	NAMA	KETERANGAN
1		*Class*	Sebagaimana kelas biasa di dalam sistem anda.
2		*Interface*	Sebuah kelas umum yang hanya menawarkan *public operations*, tapi tidak mempunyai atribut atau *method bodies*.
3		*Package*	Sebuah pengelompokkan dari kelas-kelas.
4		*Collaboration Lifeline*	Sebuah partisipan mandiri didalam sebuah interaksi. Saat bagian dan structural features mempunyai penggandaan yang lebih baik dari sebelumnya, lifelines menjelaskan hanya satu entitas interaksi.
5		*Enumeration*	Sebuah *classifier* yang termasuk dalam daftar nilai bernama berbagai jenis atribut.
6		*Node*	Sebuah *classifier* yang merepresentasikan sebuah komputasi sumber daya run-time, pada umumnya memiliki memory dan kemampuan proses yang tinggi.
7		*Datatype*	Sebuah jenis bernilai yang tidak mempunyai identitas, termasuk dalam tipe primitif fan enumeration.
8		*Artifact*	Sebuah potongan fisik dari informasi yang digunakan atau diproduksi dari sebuah proses pengembangan perangkat lunak. Informasi ini dapat berupa model, file sumber, script dan binary executable.

Tabel 2.6 Lanjutan Notasi Diagram *Class*

NO	GAMBAR	NAMA	KETERANGAN
9		*Aliased*	Sebuah jenis yang banyak digunakan dalam C++. Memungkinkan *user* untuk mengkustom sebuah jenis, seperti pointer atau refrensi didalam C++.
10		*Utility Class*	Satu set dari variabel global dan prosedur yang telah dikelompokkan didalam sebuah deklarasi kelas. Merupakan penamaan dari sebuah attribut nonmember, sebuah operation scoped dari sebuah kelas. Sebuah kegunaan kelas yang merepresentasikan sebuah jenis yang memiliki instansiasi. Sebuah attributes dan operations dari kegunaan menjadi global variabel dan prosedur.
11		*Actor*	Sebuah peran yang dikendalikan oleh *user* didalam sistem.
12		*Generalization*	Hubungan antara sebuah elemen subtype dengan elemen supertype. Elemen yang terdapat pada relasi tersebut harus mempunyai atribut yang sama.
13		*Implementation*	Sebuah implementasi hubungan antara dua elemen.
14		*Nested Link*	Sebuah hubungan antara kelas yang berbeda.
15		*Dependency*	Sebuah hubungan antara dua elemen yang definisinya bergantung terhadap elemen yang lainnya dan jika merubah salah satu makan akan merubah yang lainnya juga
16		*Association*	Hubungan yang mengasosiasikan sebuah aktor ke *use case*. Hubungan ini menunjukkan keikutsertaan aktor didalam *use case* dan hanya *link* ini yang menjadi satu-satunya penghubung antara aktor dengan *use case*.

4. Diagram *sequence*

Dalam diagram *sequence* dijelaskan secara rinci bagaimana suatu proses berjalan dalam suatu *use case*, didalamnya juga terjadi interaksi

antar kelas, operasi yang terlibat, urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi (Schmuller, 1999).

Diagram *sequence* menjelaskan aspek dinamis dari sistem yang akan dibangun. Diagram ini juga menunjukkan serangkaian pesan yang dipertukarkan oleh obyek-obyek yang melakukan suatu tugas atau aksi tertentu. Berikut beberapa notasi yang digunakan dalam diagram *sequence* terlihat dalam tabel 2.8.

Tabel 2.7 Notasi Diagram *Sequence*

NO	GAMBAR	NAMA	KETERANGAN
1		*LifeLine*	Sebuah partisipan mandiri didalam sebuah interaksi. Saat bagian dan structural features mempunyai penggandaan yang lebih baik dari sebelumnya, lifelines menjelaskan hanya satu entitas interaksi.
2		*Actor*	Sebuah peran yang dikendalikan oleh *user* didalam sistem.

Tabel 2.8 Lanjutan notasi Diagram *Sequence*

NO	GAMBAR	NAMA	KETERANGAN
3		*Synchronous Massage*	Sebuah pesan dimana objek yang mengirim pesan harus menunggu sampai mendapat jawaban untuk melanjutkan ke proses selanjutnya.
4		*Asynchronous Massage*	Sebuah pesan dimana objek yang mengirim pesan dapat melanjutkan proses tanpa harus menunggu jawaban.
5		*Create Massage*	Sebuah elemen yang digunakan untuk membuat sebuah lifeline atau instansiasi. Elemen ini adalah sebuah constructor operations.
6		*Massage to Self*	Sebuah pesan pesan dimana *object* mengirimkan pesan didalam *lifeline* yang sama.

2.2.5. Guidelines for Rappid Application Engineering (GRAPPLE)

GRAPPLE merupakan metodologi yang fleksibel dan memberikan

panduan yang jelas dalam proses pengembangan sistem. Metode ini terdiri dari lima bagian yaitu (Schmuller, 1999) :

1. *Requirement gathering*

   Pada tahap pertama yang dilakukan oleh pengembang perangkat lunak adalah mengambil informasi lengkap dari pengguna tentang sistem yang akan dibangun dengan wawancara dan kuisioner. Wawancara dilakukan langsung dengan pengguna yang menginginkan adanya sistem ini dan dengan pengguna yang berhubungan langsung dengan sistem. Tahap ini menyarankan untuk mewawancarai pengguna yang memiliki kemampuan teknis. Analisis masalah, fungsi dan kebutuhan sistem termasuk dalam tahap *requirement gathering.*

2. *Analysis*

   Di tahap *analysis* yang dilakukan adalah menggali lebih dalam hasil yang diperoleh dalam tahap sebelumnya. Tahap ini mengkaji permasalahan pengguna dan menganalisis solusinya. Pada tahap ini yang termasuk didalamnya antara lain adalah pengembangan data dan informasi dari *requirement gathering* serta pembuatan diagram.

3. *Design*

   Tahap *design* dilakukan untuk merancang solusi yang dihasilkan pada tahap *analysis* dan *design* dapat berjalan dua arah saling menyesuaikan sampai diperoleh rancangan yang tepat. Pada tahap ini yang termasuk didalamnya adalah diagram yang telah dianalisis serta dibuat rancangannya.

4. *Development*

Tahap ini ditangani oleh pengembang program untuk membangun kode program dan *user interface*. Pengujian program dan dokumentasi sistem dilakukan pada tahap ini.

5. *Deployment*

Tahap *deployment* adalah tahap pendistribusian produk yang dihasilkan kepada pengguna. Tahap ini mencakup instalasi dan perencanaan *backup* data bila diminta oleh pengguna sesuai dengan perjanjian sebelumnya.

2.2.6. Android

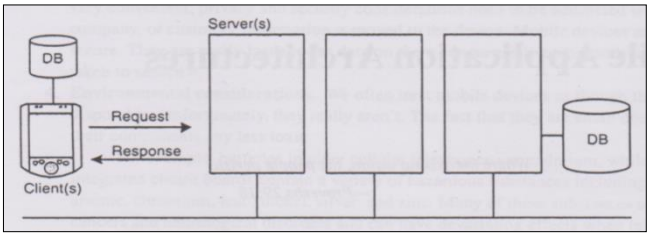
Android merupakan subset perangkat lunak untuk perangkat *mobile* yang meliputi sistem operasi, *middleware*, dan aplikasi inti yang dirilis oleh Google (Mulyadi, 2010).Sedangkan android SDK menyediakan *tools* dan API (*Application Programming Inferface*) yang diperlukan untuk mengembangkan pada *platform* android dengan menggunakan bahasa pemrograman java.

2.2.7. Arsitektur Teknologi Mobile

Arsitektur aplikasi merupakan pemodelan untuk menjelaskan atau mengilustrasikan tampilan keseluruhan dari perangkat lunak dan perangkat keras yang diperlukan untuk membangun sistem aplikasi tersebut secara keseluruhan (Lee, 2004). Pada arsitektur teknologi *mobile* terdiri dari beberapa prinsip dasar sebagai berikut:

1. Client Server

Arsitektur aplikasi biasanya dimodelkan kedalam asritektur *client-server* dimana satu atau lebih perangkat *client* melakukan *request* informasi kepada aplikasi *server* dan aplikasi *server* secara otomatis melakukan respon terhadap *request* dari *client* (Lee, 2004). Arsitektur *client-server* secara umum dapat dilihat pada gambar 2.10.



Gambar 2.1 Arsitektur *Client-server*

2. Client

Client merupakan sembarang sistem atau proses yang melakukan suatu permintaan data atau layanan ke *server* (Yuswanto, 2003).

3. Server

Server adalah komputer *database* yang berada di pusat, dimana informasinya dapat digunakan bersama-sama oleh beberapa *user* yang menjalankan aplikasi di dalam komputer lokalnya yang disebut dengan c*lient* (Yuswanto, 2003).

4. Tipe Koneksi

Ada beberapa tipe koneksi pada arsitektur teknologi *mobile,* sebagai berikut (Lee, 2004) :

a. *Always connected*

Perangkat *client* selalu terhubung kepada *server* sebagai contoh sebuah telepon seluler normalnya beroperasi pada *mode* selalu terhubung dengan operator selulernya.

b. *Partially connected*

Pada jenis koneksi ini perangkat tidak selalu terhubung dengan *server.* Sebagai contoh sebuah perangkat *mobile* kantoran yang terhubung ke *server* hanya dalam waktu-waktu tertentu atau kondisi tertentu seperti sistem absen *finger print*.

c. *Never connected*

Perangkat *mobile* yang tidak pernah terhubung kepada *back-end system* seperti perangkat *mobile gaming.*

2.2.8. Java

Java adalah bahasa pemrograman berorientasi objek. *Java* berfokus pada objek, membuat objek, melakukan manipulasi pada objek dan membuat objek-objek bekerja secara bersama-sama. *Java* mewujudkan objek dengan menggunakan kelas. Pada Java terdapat kumpulan kelas standar yang dikenal dengan *Application Programming Interface* (API) *Java*, selain itu dapat juga dideskripsikan kelas sendiri sesuai kebutuhan (Sinaga, 2005).

2.2.9. Hypertext Preprocessor (PHP)

PHP sebagai *server-side scripting language* maksudnya adalah *script* PHP dijalankan di *server*, artinya *browser* hanya menerima *output*

dari *script* PHP saat memanggil sebuah *script* PHP, dengan kata lain komputer *client* tidak akan bisa melihat kode PHP melainkan akan melihat *output* dari *script* PHP dalam bahasa HTML (Nugroho, 2004). Bahasa pemrograman PHP digunakan untuk membangun aplikasi yang berjalan pada sisi *server* yang akan berhubungan langsung dengan aplikasi yang dipasang pada telepon genggam (*client*).

2.2.10. MySQL

MySQL adalah *multiuser database* yang menggunakan bahasa *Structure Query Language* (SQL) (Sunarfrihantono, 2003). *Structured Query Language* (SQL) merupakan salah satu fasilitas yang digunakan untuk menuliskan *query* dan merupakan bahasa yang dapat digunakan untuk mendefininisikan skema basis data *Data Definition Language* (DDL), dapat juga digunakan untuk manipulasi isi dari satu atau lebih tabel yang mempunyai keterkaitan antara satu tabel dengan yang lain *Data Manipulation Language* (DML).

Selain sebagai *database server*, MySQL adalah *Relation Database Manajemen System* (RDMS) yang didistribusikan secara gratis dibawah lisensi GPL (*General Public Lisence*). Suatu *database relational* menyimpan data dalam table-tabel terpisah, hal ini memungkinkan kecepatan dan fleksibilitas.

2.2.11. Java Script Object Notation (JSON)

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah

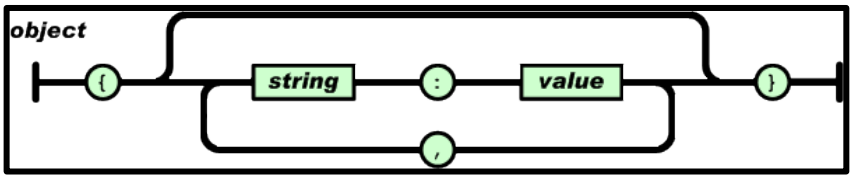
diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari bahasa pemprograman *JavaScript*, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data.

JSON terbuat dari 2 struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*)

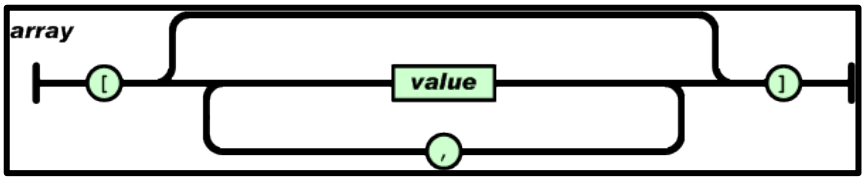
JSON menggunakan bentuk sebagai berikut:

1. Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan: (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh, (koma).



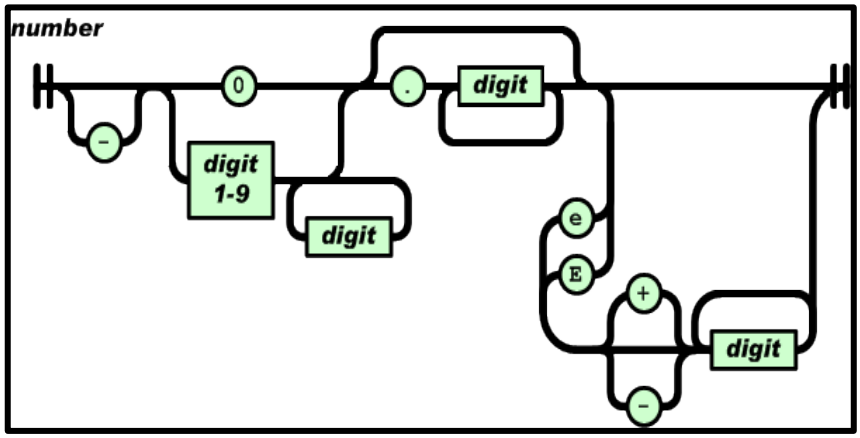
Gambar 2.2 Bentuk *Object* dari JSON

2. Larik (*Array*) adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



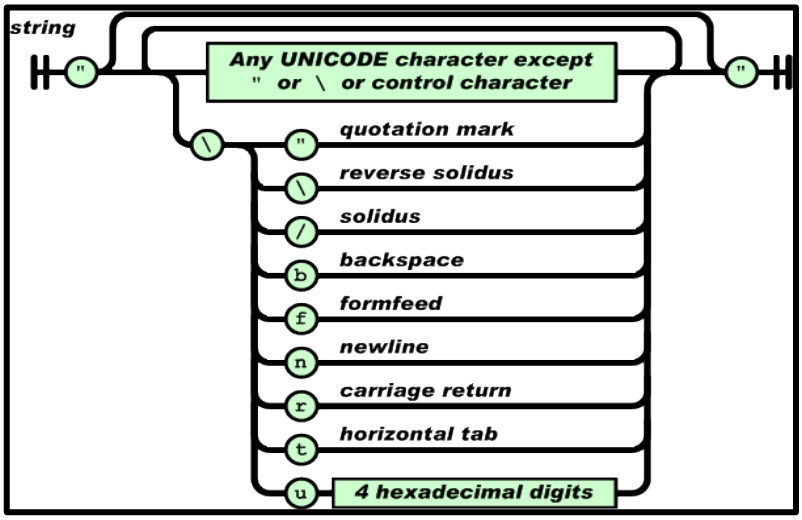
Gambar 2.3 Bentuk Larik (*Array*) dari JSON

3. Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Gambar 2.4 Bentuk Angka dari JSON

4. String adalah kumpulan dari nol atau lebih karakter Unicode, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Gambar 2.5 Bentuk *String* dari JSON

2.2.12. Posyandu(Pos Pelayanan Terpadu)

Posyandu merupakan salah satu bentuk Upaya Kesehatan Bersumberdaya Masyarakat (UKBM) yang dikelola dan diselenggarakan dari, oleh, untuk dan bersama masyarakat dalam peyelenggaraan pembangunan kesehatan, guna memberdayakan masyarakat dan memberi kemudahan kepada masyarakat dalam memperoleh pelayanan kesehatan dasar untuk mempercepat penurunan angka kematian ibu dan bayi (DepkesRI,2011).Posyandu merupakan alternatif pelayanan kesehatan yang harus dipertahankan, mengingat Posyandu memerlukan pembiayaan yang relative rendah dan dapat menjangkau target lebih luas (Yulifah & Johan,2009).